Request for Information

Benchmarking Instructions

Department of Defense

High Performance Computing Modernization Program

June 2003

High Performance Computing Modernization Program Request for Information
Benchmark Instructions

1. Background

The Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP), managed by the DoD High Performance Computing Modernization Program Office (HPCMPO), requires enhanced functionality and capability for its shared resource centers. The purpose of this request for information via these benchmark instructions is to provide the HPCMP with accurate performance information on available HPC capabilities to be considered under the HPCMP's Technology Insertion 2004 (TI-04) activity. The intent of the benchmark suite is to provide a set of program source code listings, makefiles, runtime scripts, input files, and validated results files which represents the type of computational work performed on HPCMP resources. The overall goal of the benchmark activity is to ensure that HPC systems acquired for HPCMP shared resource centers are matched to DoD HPC user requirements. Therefore, some of the benchmark requirements will configure the system as close to an operational configuration as possible. An operationally sound configuration will provide a better understanding of how the proposed HPC system will perform once installed. All HPC systems to be considered under TI-04 must provide timing and accuracy results for the distributed benchmark suite of codes. These codes are to be run individually and meet the performance requirements set forth in this document

The goal of this benchmark is to provide the best operational configuration for the sites, not the best benchmark configuration. Vendors are strongly encouraged to document how their proposed configuration can be used operationally and are required to document where they have configured the system differently for this benchmark.

No longer will application tests be required on systems with specific numbers of processors. Instead, we have established a DoD standard performance requirement, in terms of a time-to-solution, for each application test. Specifically, each application has at one input data test case called the "standard" case, and most have a second case called the "large" case. Each test case has an associated target time obtained by running that case on the Government's IBM POWER4 SP at the NAVO MSRC. A vendor should supply elapsed wall times for each test case on three distinct processor counts. The ratio of the maximum processor count to the minimum processor count must be at least two. At least one of the three reported times must be from an actual run on some model of the proposed hardware, or a closely related system; the remaining two reported times may be estimated. The same degree of reliability will be attached to estimated times as to actual times. At least one of the three reported times must be less than or equal to one half of the Government-specified target time (i.e., "twice as fast"). Vendors are cautioned, particularly if this time is extrapolated, that they will have to demonstrate the reported level of performance if their equipment is installed. Finally, it is important that the runs supporting the actual reported times follow the Government's output convention (see section 4.2 for details).

Vendors are required to provide timing output and configuration output in a common way to speed analysis and comparison. This common output is described in Appendix A.

2. Some Changes from Previous years

This year some changes have been made to the benchmark process and codes:

1. A new synthetic code, MEMBench, will replace last years MEMBench and MEMBench2.

2. OSBench is reduced to a maximum of 64 processors.

3. Some application benchmarks have been replaced with other codes that are more representative of the required workload for TI-04.

3. Operational Instructions

The vendor must abide by all processing instructions. Any deviation, questions of interpretation, and/or proposed changes must be formally clarified and approved with the DoD HPCMPO evaluation team in writing prior to running the benchmark and submitting results. Any results submitted which do not follow the operational instructions will not be evaluated.

3.1 Benchmark Structure

The benchmark is divided into 2 parts:

1. Hardware performance and system tests (synthetic tests)
2. Application tests

3.1.1 Synthetic Performance Tests

These tests are to be run one time with a standard scheduler with no changes to the default priorities used by the scheduler. Special rules apply to the I/O tests.

3.1.2 Application Tests

The application tests are multiple test cases using a suite of codes that are to be run using the standard system scheduler with no changes to the default priorities used by the scheduler with a defined range of performance for application codes (see definitions in section 5.2.5 for each code) on a dedicated machine, i.e., with a minimal number of processes running besides the application. Please see section Appendix A for presentation of results.

3.1.3 Allowed Changes

3.1.3.1 Source Code Changes

Vendors are only allowed to change the source code to the extent needed to get the program to execute and provide correct output. If a vendor wants to submit additional runs with source code

modifications in order to improve performance, they will be evaluated. The improved performance will be scored if the evaluation shows the improvement can be implemented in the actual code. The vendor must provide timing for both the modified source code and the original source code.

All source code changes, including those allowed below, must be fully documented. All software changes become the property of the DoD HPCMPO / government and may be incorporated into and used within existing codes.

3.1.3.2 Makefile

Makefiles may be changed to allow:

1. Vendors to add libraries during the build process.
2. Vendors to add compiler option(s) for each code, but only 1 version of each compiler (C, C++, and F90) may be used. Use of multiple versions of compilers is prohibited.
3. Vendors are allowed to change the definition and location of the compilers. For example: CC=/opt/bin/cc.
4. The rules: 1, 2, & 3 above also apply to linker flags. Those changes are stated in the application sections (see section 5.2.4 used to build the executable on various platforms).

3.1.3.3 Run script changes

Scripts may not be changed except for those changes necessary to execute the code. Examples of such changes may include modifying the path names of variables, changing the number of CPUs, and the addition of environment variables. All changes to the script must be POSIX compliant.

The vendor MUST provide detailed documentation on any changes to the run scripts, documenting changes and reasons why the changes were made.

3.1.3.4 Documenting Changes

All changes must be documented, including allowed changes. The documentation must describe the rationale for each change, and for each compiler option added, detailed documentation must be provided. Documentation for use of environment variables must be detailed, and the rationale for the addition must be provided.

3.2 Configuration Disk Requirements

As the system is to be configured as close as possible to an operational system, the following will be required as part of the disk configuration and will be evaluated.

3.2.1 Configuration for the I/O tests

The I/O test does not need to be run on the configured system, but can be run on a separate node on an external system (same system type). This is suggested given the expected run time.

The IO test is benchmarking the file system; therefore, the vendor must run the benchmark on a fixed configuration. The benchmark can, however, be run a second time on a different

configuration, with the stipulation that the second configuration must be described in detail, including the cost differential between the two systems.

Before running the disk I/O test suite, the file system being used is to be created using 8 Fibre Channel RAID devices, volume manager, file system, and/or other appropriate configuration settings. These settings are to be configured only 1 time after booting the system and are not to be changed at any time during the benchmark. The I/O test can be run on a separate system from the computational systems, but that system must be part of a node that will be provided as part of the vendor computational configuration.

For example, if the vendor runs the I/O benchmark on a 4-processor system with 8 PCI buses, that configuration must be used within the configuration of the computational system. The configuration requirement is at least one-fourth of the nodes must have 4 processors with 8 PCI buses.

4. Preparing the System Timing Tables:

4.1 Benchmark Timing Tables

Timing tables in Appendix A and provided in softcopy are to be filled out and returned as part of the benchmark submission for each benchmark type. Document description is provided in Appendix A.

4.1.1 Synthetic Test Benchmark

Please provide the output file for each test. All tests must be run with the standard vendor-provided scheduler. Use of the real-time scheduler or modification to the batch scheduler is NOT ALLOWED.

4.1.2 Applications Benchmark

The timing tables as described in Appendix A must be completed in the softcopy format provided (see Appendix A). Table entries are required for each of the application runs for each CPU count.

4.2 Output Requirements

The output requirements for the application benchmarks will be defined individually; see section 5.2 and Appendix A. This includes, but is not limited to, output files to check for correctness, timing data, etc. Furthermore, a single text file will serve as a summary table of results for the application benchmark results. These files will be named:

./ded/README.results

As noted above, each job must print out timing information and the information per Appendix A must be completed. The vendor must also provide the output for the run so the HPCMPO can verify the accuracy of the data.

Vendors are welcome to provide other data regarding the performance of jobs if they so choose.

4.3 Information to Submit with Results

Three types of information are required to be provided:

1. Configuration certification
2. Hardware configuration and settings
3. Software configuration and settings

4.3.1 Configuration Certification

Any set of hardware and software bid / benchmarked must include a certification from the vendor that the configuration meets the following criteria and answers the following questions:

1. Hardware configuration is in the vendor's list of commercially available products
2. Software configuration is in the vendor's list of commercially available products
3. Hardware and software configuration have been tested together which includes:
    a. Microcode
    b. Operating system
    c. Device drivers for network cards and HBAs
    d. Device drivers for fibre channel adapters
    e. Scheduler
4. Vendor agrees to take total responsibility including providing development resources to provide software and microcode fixes for deficiencies found once installed.

4.3.2 Software Configuration

The computer systems software must be the same for each system. Only released, commercially available, and supported products are acceptable for this test or products that will be released within six (6) months of the benchmark.

The released software requirement includes, but is not limited to:

1. Operating system
2. Network drivers
3. Network stacks
4. I/O drivers (e.g.. FC-Fabric, SCSI)
5. File system software and/or volume manager
6. Compiler and libraries including I/O libraries, and MPI

7. All patches or bug fixes
8. Any additional software used as part of the benchmark configuration

Suppliers shall not be permitted to change software configuration of any system while executing this benchmark suite.

4.3.3 Hardware Configuration

Each configured system shall be documented with the hardware used. Only released, commercially available, and supported products are acceptable for this test or products that will be released within six (6) months of the benchmark. This includes, but is not limited to:

1. Memory boards, sections, and/or banks
2. Memory size
3. CPU speed and boards
4. I/O boards
5. Bus speed, both local buses and external buses
6. HBAs including firmware
7. Network adapters including firmware
8. All communications hardware including private channels
9. RAID hardware including: disks, cache, firmware, channels, GBICs, and interfaces
10. Any fibre channel switches, if used
11. Any other hardware used as part of the benchmark configuration

A table is provided in Appendix A that must be completed and provided in both hardcopy and softcopy.

4.3.4 Hardware Configuration and Settings

Vendor shall use the proposed system(s) with a configuration of their choosing to execute the benchmark suite, and the delivered system must meet or exceed the performance of the benchmarked system.

A table is provided in Appendix A that must be completed and provided in both hardcopy and softcopy.

4.3.5 Software Configuration and Settings

Since no software differences are allowed between machines or between runs, this table only needs to be completed once for all of the configurations. Any additional information that pertains to the software configuration should be provided.

A table is provided in Appendix A that must be completed and provided in both hardcopy and softcopy.

4.4 Submitting Results

The offeror will provide CD-ROMs in machine-readable format with the submission for:

1. All released software used for benchmark tests
2. Patches used in configuration for the benchmark tests
3. Instructions for building and installing the test configuration(s)
4. Results in templates provided in Appendix A for each table in softcopy file

From these CD-ROMs and with provided instructions, the DoD HPCMPO should be able to reload and rebuild the system(s).

4.5 System Configuration Changes

The System Configuration Table in Appendix A must be completed to document any and all changes made to the system to run the benchmark. If the vendor reloads the system, only these configuration changes will be applied to the system to allow it to run the benchmark. This includes, but is not limited to:

1. Disk format settings
2. Disk configuration settings
3. Kernel cache settings (e.g. name cache, disk cache)
4. Kernel configuration settings (e.g. physical I/O settings)
5. Network stack settings
6. Volume manager and file system layout and configuration
7. Any other system tunables or settings that are used as part of the benchmark

The DoD HPCMPO reserves the right to reject the offeror's proposal and require a re-test of the offeror's system for any undocumented changes. The DoD HPCMPO also reserves the right to require a Government-attended demonstration at a mutually agreeable time.

4.6 Detailed Hardware Description

The vendors need to provide the following information for each system proposed, as described in Appendix A:

1. Processor MFLOPs
2. Number of floating point units and types
3. Number of integer units and number of adders
4. Cache size used and cache sizes available
5. Processor speed (MHz)
6. Memory speed

7. Memory bandwidth or bandwidths for each node type

8. Disk bandwidth

9. Type of architecture (hypercube, torus, etc).

4.7 Additional Required Documentation

The offeror shall provide the following information with the proposal:

1. Completed System Timing Table for each system in the initial target configuration, signed and certified by the offeror's representative to be true, complete, and accurate. A computer file containing a computer-readable copy of each table shall also be provided.
2. Completed Benchmark Summary Table, signed and certified by the offeror's representative to be true, complete, and accurate. A computer file containing a computer-readable copy of each table shall also be provided.
3. Documentation of the application source code changes as follows:
   - Listing of each routine changed with all changes clearly marked
   - Rationale for each change made
   - Comparison of timing runs with and without the change(s)
4. Documentation of all *makefile* changes as follows:
   - Listing of each *makefile* changed with all changes clearly marked
   - Rationale for each change made
5. Documentation of all run script changes as follows:
   - Listing of each run script changed with all changes clearly marked
   - Rationale for each change made

4.8 Use, Copying, and Transfer Restrictions

The benchmark suite provided by the government / HPCMPO for this program shall be restricted for use in responses. The use of this benchmark suite for any other purpose is not authorized. Some of these benchmark codes are subject to export or licensing restrictions. Signed agreements shall be completed and returned to the Contracting Officer at the address below before these codes are released by the government, and recipients shall not further release these codes. The reproduction or distribution of this benchmark suite to any organization other than the originally intended recipient shall not be authorized except for direct use in responding. Offerors selected for award shall retain these materials for use during acceptance testing and for measuring upgrade options.

Create three copies of the results and send one copy to each of the below addresses.

DoD High Performance Computing Office
Attn: Contracting Officer
1110 N. Glebe Rd., Suite 650
Arlington VA 22201
(703) 812-8205

Instrumental Inc
Attn: Kim Payne
2748 E 82nd St
Bloomington, MN 55425-1365
(952) 345-2822

William A. Ward Jr.
C/o Ms. Daffney Wells
Computer Sciences Corp
3530 Manor Drive, Suite 4
Vicksburg, MS 39180
(601) 634-2512

5. Benchmark Description, Requirements, and Resource Estimates

5.1 Synthetic Codes

The following codes in this section must be run with one scheduler as documented in section 2.1.1.

- I/O tests
    - IO tests are designed to test the file system; therefore, a large amount of data must be written and fragmented. To ensure that the data does not stay resident in system memory (RAM), large files in excess of 80 MB must be read and written.
- OS tests
    - The OS tests are designed to test the basic OS functions such as TCP stack and file operations, while scaling up to the maximum number of processors on a node.
- Memory tests
    - The memory test is designed to determine the memory bandwidth available for processing as well as make a comparison of this bandwidth to bandwidth received when performing CPU intensive operations.
- Network tests
    - The network tests are designed to determine the interconnect bandwidth between processors while performing different interprocessor communication functions such as scatter/gather, and allreduce.
- CPU tests
    - The CPU tests are designed to determine the CPU performance on different mathematical functions. (See section 5.1.5.1.)

Hardware Requirements to Run Synthetic tests

Test	Processors Required	Memory Required	Comments
CPUBench Series 100 - 300	1	Minimal amount outside cache	
CPUBench Series 500, 600, 700	16 to 512	64K to 4 MB per processor	

NETBench	16 to 512 processors	Minimal	
MEMBench	128	~ 512 MB per processor, is required	Recommend exactly 128 processors
OSBench	Maximum processors in a node, up to 64 total	Minimal	Runs on one node
IOBench	1	Minimal	Tests file system; only 1 processor is required; but requires ~4TB of disk.

The point of contact for all questions on these benchmarks is:

Robert Graham
Instrumental, Inc
605-483-3295
rgraham@instrumental.com

5.1.1 I/O Tests

Since the objective of the I/O tests is to benchmark the file system, a large amount of data is written out to disk in the form of large files. This will ensure the data is forced out of main system memory (RAM) – a particular concern when benchmarking architectures with large shared memory. As a consequence, the overall data set is very large, and the test can therefore, take an extended amount of time to run. Additional information about the I/O tests can be found in Sections 3.2 and 3.2.1.

5.1.1.1 Descriptions

The I/O benchmark must run on the following configuration:

- 8 Fibre channel RAID controllers
- RAID disks of at least 72 GB
- RAID configuration of 8+1 RAID-5
- File system size of at approximately 4.3 TB
- Server configuration with at least 8 HBAs each connected to a switch or direct connected. No more than one LUN should be connected to each HBA.

The I/O load provided mimics the I/O requirements for a shared resource center site's I/O workloads. The tests are described in the sections below. The test process is:

- Generation of a 2.5 TB file.

- Generation 2 million files, then removal of ½ of files (which will fragment the file system).

- Run the Multiple Stream test, which will use the data from the above two tests to determine I/O performance.

Fragmentation of the file system is being done as all sites have seen operational problems with large file systems.

Note: As mentioned earlier in this document, tunable changes allowed include changes to the script for preallocation and other command based tunables. No code changes are allowed. Any script changes must be documented.

5.1.1.2 2.5 TB Test

For this test, 16 MB I/O requests will be written to a single file that is 2.5 TB in size. This test will be timed. As stated above, no code changes are allowed. The data from this test will be utilized in the Multiple Stream test described in section 5.1.1.4.

5.1.1.3 Fragmentation

Over two million files of size 32 KB will be written with a 32 KB request size. These files will be written in 2000 directories with 1000 files per directory. The total data size is 64 GB. After the files are written, ½ of the files will be removed. This test will be timed. As stated above, no code changes are allowed. The data from this test will be utilized in the Multiple Stream test described in section 5.1.1.4.

5.1.1.4 Multiple Stream test

This test utilizes the data created in sections 5.1.1.2 and 5.1.1.3 above. For this test, the tests in the table below will be running in parallel. All of the following are to be run on the same file system. Output from the test is total wall time and CPU, user and system time. As stated above, no code changes are allowed.

Tests	**Requests Sizes**
Write single 50 GB file forward; read forward	64K
Write 200 GB file forward; read forward	1MB
Write 10 MB file backward; read backward	64K
Write 10 MB random read/write-modify; random read	64K
Read part of the 2.5 TB file seek 200 GB read 100 GB	1MB

5.1.1.5 Running the tests

All tests are run from a single interactive script and can be run in the background. The script *do_io* is to be run from the file system that was *mkfs*'ed and mounted which is called /usr/tmp/dodbench.

The following must be in your default path:

- Compiler
- Default command such as make

Before performing a make on the source code, ensure that the proper flags are included in the makefile for your platform. These flags must have 64-bit options as well.

5.1.1.6 Running the tests independently

The programs iobench, mktree and rmtree can be run independently of the script mentioned above. The usage messages from the programs are provided below:

IOBENCH

$ iobench –h

Usage: iobench [-h] [-r report_file] -s file_size(k,m,g) –b block_size(k,m,g) -i operation [-S stride(k,m,g)] data_file

Operations are:

sw: sequential write

bw: backward write

sr: sequential read

rr: random read

br: backward read

rw: random read_write

tw: test write

MKTREE

mktree creates a directory with 1000 files under it. The script do_io controls the number of directories. The arguments for mktree are "file_system" and "index." "File_system" is where you want the directory to be created and the "index" value is used to name the directory. The directory name will have the format "0-index-0."

$./mktree

Usage: mktree file_system index

RMTREE

rmtree removes a directory with 1000 files under it created by mktree. The script do_io controls the number of directories. The arguments for rmtree are "file_system" and "index." "File_system" is where the directory to be removed is located, and the "index" value points to a directory name. The directory name is in the format "0-index-0."

./rmtree

Usage: rmtree file_system index

5.1.1.7 Results

Results will be put in iobench3/results. Please provide that file with your submission.

5.1.2 Operating System Tests

The OS benchmark tests the ability of the OS to transfer data and perform file operations. These operations are required functions at the MSRCs. This test runs quickly on some architectures, but is time consuming on others. This test was reduced as far as possible (in TI-03) to still receive valuable results. This test was not changed in TI-04, except to limit the maximum number of processors to 64.

5.1.2.1 Descriptions

The Operating System (OS) tests will measure the performance of the following OS functions:

- System Calls
- Interprocess Communication
- TCP Scalability

To run the tests see paragraph 5.1.2.5 Running the tests, below.

Maximum number of processes for these tests is 64. This test runs on one node of the computer system. The vendor should run on the maximum of 64 processors, or the number of processors on a node.

5.1.2.1.1 System Calls

For this test, we will perform a loop that performs the following tests. The loop will perform several sets of iterations, starting at 64 thousand (64*1024), and doubling until the max value of processors is reached. For example, for 1 processor, 64K iterations would be run. For 2 processors, a new process will be forked and 64K iterations will be run out of both the parent and child, for a total of 128K. For 4 processors, 4 processes will run a total of 256K iterations. Maximum number of processors is 64. Each set will be timed independently, and results will be put into the results folder, in a file named results_syscl.

Results are printed in clock ticks, which is the number of clocks per second, usually 100. This program uses the open, close, read and write commands. See section 5.1.2.5 below for details on running this test.

Test	Request Sizes
Open a file	
Write 1 integer number to the file	1 integer
Close the file	
Open the file	
Read 1 integer from the file	1 integer
Close the file	

5.1.2.2 Interprocess Communication

For this test, we will open a UNIX pipe between two processes, and transfer 2 GB of data in different block sizes. The program creates a new process using the fork command. The child that was forked writes to the pipe, and the parent reads from the pipe. The data is read in block sizes as shown in the table below.

The program will perform several iterations of the test. Each iteration contains four (4) 5 GB transfers using the block sizes in the table. The number of iterations is based on the number of processors in the Makefile. The results are placed in the results folder in a file named results_pipe. The results are reported in power of 2 intervals. For example, for a 16-processor input, results would be reported for 1, 2, 4, 8, and 16 processes. The results will also check one number that is transferred through the pipe to verify the correct number is being received.

Results are printed in clock ticks, which is the number of clocks per second. A tick is usually defined as .01 second. This program uses the pipe(), read and write commands. See section 5.1.2.5 below for details on running this test.

Test	Block Sizes
Open pipe, write 2 GB	512, 1024, 2048, 4096

5.1.2.3 TCP Scalability Test

For this test, we create a TCP socket, and transfer data through the socket in different sizes. The test opens the socket and transfers 128 MB of data through the socket. This test utilizes the socketpair, send and receive commands. The fork command is used to create a new process. The parent process receives the data the child process sends.

The program will perform several iterations of the test. Each iteration contains four (4) 2 GB transfers using the transfer sizes in the table. The number of iterations is based on the number of processors in the Makefile. The results are placed in the results folder in a file named results_tcpsc.

The results are reported in power of 2 intervals. For example, for a 16-processor input, results would be reported for 1, 2, 4, 8, and 16 processes.

Tests	Requests Sizes
Write data through a TCP socket and read data on other end	4K, 16K, 32K, 64K

Results are printed in clock ticks, which is the number of clocks per second, usually 100. This program uses the socketpair, send and receive commands. See section 5.1.2.5 below for details on running this test.

ANSI C compiler, POSIX compliant, is required.

5.1.2.4 Resource Estimates

Each iteration of the tests (syscl, pipe & tcpsc) can take over 5 minutes, and this time is doubled for each iteration, so a 16 processor test would run 5 times (2 to the 5th), and would take 25 minutes assuming perfect scaling of processes to 16 processors. The full test will probably take much longer than this.

5.1.2.5 Running the tests

All tests are run from a single interactive script and can be run in the background. The script *do_os* is to be run. The script will create a directory named osbench. YOU MUST UPDATE THIS SCRIPT!!! The do_os script has one variable at the top, on line 3. The variables must be set to match your system configuration. The variable PROC_PER_NODE is the total number of processors in the computer node. This number determines how many processes to create when running the benchmark.

The following must be in your default path:

- Compiler
- Default command such as make

In the Makefile, the user may add any source, library, or compiler flags in their section. If the vendor does not have a section, they may make a section. (Each vendor has a section for flags, all are commented out).

make clean will remove all object, results, and core files, and leave the *.c files, Makefile and do_os command file. This is used to rerun all the tests.

make clobber will remove all files including c source files and results files. It will also remove the osbench directory.

make tar will make a tar file with all the files in the osbench directory, including the results directory.

To run a program independently, type:

make progname.c

This will create the object program.

To run syscl, pipe, or tcpsc, you will need to add the number of processors, i.e.

syscl 16

This would run the syscl program with 16 processors.

5.1.2.6 Results

There will be three results files, one for each of the tests. Results will be put in osbench/results. These files will be the results of the operating system benchmarks.

5.1.2.6.1 Results_syscl

The results of the system call benchmark will have several sections, each section for a set of iterations. The number of processors selected is printed, and the size of an integer. A test number is printed out to verify the same number written was read. The program will print out a time for every process that is run. If 64 processors are input, the final step will have 64 sets of results. The clock ticks will be printed out, normally 100. The results are in clock ticks. The number we are interested in is the diff. This is the benchmark time in clock ticks of a second.

5.1.2.6.2 Results_pipe

The results of the pipe benchmark state the number of processors used, then several sets of iterations. The iterations have four sets of timings, one for each block size listed in the pipe section table above. Each set of timings will have a time for every process that was run. If 64 processes were input, the final set of timings will have 64 processes. Larger block sizes should have faster times (smaller numbers). The times are in clock ticks, which is usually 100ths of a second. Divide the times by 100 to get seconds.

5.1.2.6.3 Results_tcpsc

The results of the TCP transfer benchmark state the number of processors used, and then several sets of iteration times. Each of the iterations contains four timings. The timings are for the different transfer sizes listed in the tcpsc section table above. Each set of timings will have a time for every process that was run. If 64 processes were input, the final set of timings will have 64 processes. The larger transfer sizes should have faster times (smaller numbers). The times are in clock ticks, which is usually 100ths of a second. Divide the times by 100 to get seconds.

5.1.3 Memory Tests

The purpose of the memory test is to test the maximum bandwidth the processor can obtain from different levels of the memory system. There are two tests; one test performs loads and stores across large data sets. The other test runs a processing function across a large data set.

It is strongly recommended that this test be run on a system with exactly 128 processors. If a vendor runs on a system with greater than 128 processors, the vendor cannot disable processors to achieve faster performance, unless that is a configuration they are bidding. The vendor must guarantee the benchmarks will obtain the same results on a system with only 128 processors.

5.1.3.1 Descriptions

There are two components of the memory test, a load and store test, and a processing test. They are named MEMBench Load and Store (MEMBench.LS) and MEMBench Processing (MEMBench.P). The load and store test will perform loads and stores across several large data set sizes. The processing test will perform processing function across several large data set sizes.

MEMBench.P is coded in both OpenMP and MPI. The vendor may use whichever code provides the best performance. The MEMBench.P data set is COMPLEX*8, (128 bit complex words).

MEMBench.LS is written in MPI.

5.1.3.1.1 MEMBench.LS

The memory hierarchy performance is being tested as memory usage scales from 1 processor to 128 processors. The test is written using MPI, and OPEN MP.

The memory benchmark is designed to achieve an understanding of the memory system performance. The test will run automatically, but will cycle through several tests with several data set sizes.

MEMBench Load and store test	
Test	Data Set size
1	(2K x 2K) 64 MB
2	(4K x 4K) 256 MB
3	(8K x 8K) 1 GB
4	(16K x 16K) 4 GB
5	(32K x 32K) 16 GB

MEMBench.LS will run on 1, 2, 4, 8, 16, 32, 64, and 128 processors. The benchmark will run automatically from the run script.

5.1.3.1.2 MEMBench.P

This performs a 2D FFT across several data sets, and several sets of processors. The memory hierarchy performance is being tested as memory usage scales from 1 processor to 128 processors, utilizing a large portion of the memory on a node. The test performs a 2D FFT, which involves an FFT, a transpose, and another FFT. The test is written using MPI, and OPEN MP. Some of the characteristics of the test are as follows:

1. The data is complex *16 (128 bit complex words)

2. The test runs five (5) different data set sizes, but all the data set sizes are square. The data set sizes are listed in the table below.

3. The test runs all five data set sizes on different numbers of processors, starting at 1 and doubling each time, up to 128 processors.

4. The program will automatically halt when there is not enough memory available, for example if your computer has 1 GB per processor, if the vendor is running on 1 processor, and the data set size is larger than 1 GB, the processing will halt and the processing on the next larger set of

processors will begin. Vendors will not be punished if codes do not run on processors with more than 1 GB.

The vendor is free to optimize this code by inserting library routines. But, we do require the time for the FFT and for the transpose independently (as in the code provided). These library routines must be released versions, please include the release number with your submission. Also, the vendor must be able to compute the correct answer for the three routines. Overly optimistic benchmark results must be accompanied by an explanation of how the computer is able to achieve high efficiency in those routines.

This FFT does not perform a comparison of results, but a comparison may be performed if their machine is purchased. Vendor guarantees the accuracy of the FFT, while achieving the reported benchmark time.

This program will run programs on data set sizes of

Processing test	
Test	Data Set size
1	(2K x 2K) 64 MB
2	(4K x 4K) 256 MB
3	(8K x 8K) 1 GB
4	(16K x 16K) 4 GB
5	(32K x 32K) 16 GB

MEMBench.P will run on 1, 2, 4, 8, 16, 32, 64, and 128 processors. The benchmark will run automatically from the run script.

As mentioned above, inserting library calls or library routines for these algorithms may optimize these three routines. The vendor can only use released library calls. The algorithms must run on the supplied data set.

The transpose (CTM) is an "in place" algorithm. The Vendor may use an "out of place" algorithm, providing the data integrity is maintained. If the vendor uses an out of place algorithm, they must state this in the response.

5.1.3.1.3 Requirements

A Fortran compiler is required to run these tests. Also, the benchmark requires 512 MB of memory per processor.

5.1.3.1.4 Resource Estimates

The entire benchmark will run in approximately three hours, depending on the architecture and the amount of RAM.

5.1.3.1.5 Running the tests

For more details, please read the README file in the membench directory.

When MEMBench.tar is extracted, there will be a directory created called membench. Change directory to the MEMBench. In the makefile you will find sections for Linux, Cray, IBM, SGI, Compaq, and Sun compiler flags and library search paths. Uncomment the section which is

appropriate and edit it for optimal performance on your platform, or if necessary add a new section. Type "make all" to make the programs.

There are three scripts to run different versions of MEMBench. Vendors are required to run the Load and Store test, and are required to run one version of the processing test.

1. run_mem1.ll (run F90 load-store tests for 1,2, 4, 8, 16, 32, 64, 128 CPUs)

2. run_mem3.ll (run F90 OpenMP SMP 2-D FFT for 1,2, 4, 8, 16, 32, 64, 128 CPUs)

3. run_mem4.ll (run F90 MPI 2-D FFT for 1, 2 4, 8, 16, 32, 64, 128 CPUs)

After you have run the tests, type 'make tar' in the membench directory. This will create a tar file named 'membench.tar'. This is what is to be returned. This tar file will contain the results and the source files, which are to be returned.

5.1.3.1.6 Results

The output of the tests is written into the 'results' directory. There will be sixteen output files with results. Each processor count will have an output file, and there are two sets of outputs.

The MEMBench.LS test will contain the following information.

- The wall clock time

MEMBench.P test will contain the following information for the FFT, the Transpose, and for the overall time.

- The wall clock time for FFT
- The wall clock time for the transpose
- The wall clock time for the entire test

5.1.4 Network Tests

The network tests are designed to test the interconnect between CPUs and nodes. Since most of the codes running at MSRCs require a large number of processors, this test is important for determining system performance.

The NETWORK synthetic benchmark consists of the following five Message Passing Interface (MPI) tests:

- A Point-to-Point Blocking test
- A Point-to-Point Non-blocking test
- A Point-to-Point Persistent Non-blocking test
- A Broadcast test

- An *Allreduce* test

5.1.4.1 Description

What follows is a description of these tests, followed by a description of how to run these tests.

5.1.4.1.1 Point-to-Point Blocking Test

This test uses a master-slave approach. The master sends a sequence of messages consisting of n real*8 numbers to the slave. Then when the slave has received all the messages, the slave sends back a one-word reply. Both the sends and receives are blocking. This test failed on some of the centers platforms until we set the MPI environment variable MPI_MSGS_PER_PROC to a number greater than 4096, which is the maximum number of messages sent. This problem occurred for small messages, because the vendor's implementation of MPI_SEND placed the message in the header, and was not really a blocking send. All the messages sent are a power of 2 number of real*8 numbers starting at 2. The program runs on from 16 to 512 processor subsets. For an example of 16 processors, it uses a single master, and sequences through 15 slaves, one at a time, and records the minimum and maximum MBytes rate for each message size. This technique will show performance variations because of asymmetries in the network.

5.1.4.1.2 Point-to-Point Non-blocking Test

Application programmers usually prefer to use non-blocking message passing in their programs. All the messages sent are a power of 2 number of real*8 numbers starting at 2. The program runs on from 16 to 512 processor subsets. For an example of 16 processors, it uses a single master, and sequences through 15 slaves, one at a time, and records the minimum and maximum Mbytes rate for each message size. This technique will show up performance variations because of asymmetries in the network.

5.1.4.1.3 Point-to-Point Persistent Non-blocking Test

In many applications, the same message patterns are repeated over and over, such as a time step loop in a transport code. Because of the overhead in setting up a communication, MPI provides a "persistent" methodology to reduce the start-up time in communication. This test attempts to quantify the savings, and performs the same message passing as the previous test, but uses the "persistent" interface routines.

All communication is non-blocking. All the messages sent are a power of 2 number of real*8 numbers starting at 2. The program runs on a from 16 to 512 processor subsets. For an example of a 16 processors it uses a single master, and sequences through 15 slaves, one at a time, and records the minimum and maximum MBytes rate for each message size. This technique will show up performance variations because of asymmetries in the network.

5.1.4.1.4 Broadcast Test

This test performs a broadcast of a sequence of messages with a size of a power of two of real*8 numbers starting at 2. It uses the processor with MPI rank 0 as the root. The test uses all available processors. In this test the minimum and maximum MBytes rates are of course the same, since a master-slave strategy is not used as in the previous tests.

5.1.4.1.5 AllReduce Test

This test performs an allreduce of a set of local arrays whose size is a power of 2 real*8 numbers starting at 2. It uses the processor with MPI rank 0 as the root. The test uses all available processors. In this test the minimum and maximum MBytes rates are of course the same, since a master-slave strategy is not used as in the previous tests.

5.1.4.2 Requirements

ANSI C compiler, POSIX compliant, and MPI support is required to run these tests

5.1.4.3 Resource Estimates

All the tests may require 256 MBytes of memory for each processor. The message-passing tests may require a search path to link the MPI library. The Fortran 90 coded tests use the POSIX standard wall-clock timing routine "get_system_clock" and the C coded tests use the C-callable routine "get_time_of_day." Netbench will require about 30 minutes to run.

5.1.4.4 Running the tests

In the Makefile you will find sections for Linux, Cray, IBM, SGI, Compaq, and Sun compiler flags and library search paths. Uncomment the section which is appropriate and edit it for optimal performance on your platform, or if necessary add a new section. If you use GNU's make (gmake), edit the appropriate section of the file "cmplrflags.mk." To make all of the netbench test enter the command: make. You will find a subdirectory of netbench called scripts. It contains two simple interactive run scripts called "do_net1", and "do_net2." You will have to edit both scripts to edit the line, which does a cd to the working directory, and possibly redefine the environment variable RUNCMD, which invokes a MPI run on your platform.

The first script "do_net1" will create a subdirectory called "results" (if it does not exist), and run the Fortran 90 coded tests. The second script "do_net2" will execute the C coded tests. After you have run all the tests, enter the command: make tar. This will create a tar file called "netbench.tar", which is the deliverable for netbench. It will contain the source code and the results files, which are used to evaluate the benchmark.

5.1.4.5 Results

All tests (both Fortran 90 and C) report a size (called n), and minimum and maximum Mbytes count. The minimum and maximum will possibly be different for the point-to-point tests, but should be identical for the collective tests (broadcast and allreduce).

5.1.5 CPU Tests

The CPU tests will characterize the performance of the CPU in several modes, yielding an assessment of both CPU and compiler performance and revealing compiler deficiencies. Since the MSRCs use the systems in a production environment, a stable compiler is a requirement.

The CPU tests examine 19 fundamental computation kernels, comparing their performance with respect to precision (32-bits versus 64-bits), language (C versus F90), and cache effects (cache-friendly versus cache-unfriendly). There is also a BLAS benchmark, and three benchmarks that require more than one processor, scaling up to a maximum of 512 processors.

5.1.5.1 Descriptions

The CPU synthetic benchmark tests consist of the following:

- 19 simple kernels coded in Fortran 90 and C which have been coded for maximal cache re-use

- 19 simple kernels coded in Fortran 90 and C which have been coded for minimal cache re-use
- The BLAS routines SAXPY, DAXPY, SGEMV, DGEMV, SGEMM, DGEMM
- The parallel ScaLAPACK Gaussian Elimination Solver PDGESV
- The parallel ScaLAPACK routine PDGEHRD, which reduces a non-symmetric matrix to upper Hessenberg form for eigenvalue calculations
- A "C" coded parallel Conjugate Solver of Laplace's Equation

What follows is a description of these tests, followed by a description of how to run these tests.

5.1.5.1.1 Cache-Friendly Fortran 90 and C Kernel Tests

The 19 tests contained in this suite are derived from simple kernels devised by the PARKBENCH Committee headed by Roger Hockney in 1994. We have coded these simple loops in both Fortran 90 and C in such a way that allows maximal cache re-use. These loops are tested for vectors with length of a power of 2 starting at 2. These tests are performed in 32-bit and 64-bit precision in both Fortran 90 and C, for a total of 68 tests.

5.1.5.1.2 Cache-Unfriendly Fortran 90 and C Kernel Tests

The 19 tests contained in this suite are a modification of the first suite, which minimizes cache re-use by adding another dimension to the data so that new data is read and written at each stage. These loops are tested for vectors with length of a power of 2 starting at 2 as well. On cache-based systems there is an obvious degradation in performance between the cache-friendly suite and cache-unfriendly suite. These tests are also performed in 32-bit and 64-bit precision in both Fortran 90 and C, for a total of 68 tests.

5.1.5.1.3 Native BLAS Tests

This suite tests the performance of the most important Basic Linear Algebra Subroutines (BLAS), which are commonly used in LAPACK and ScaLAPACK. The first level BLAS routine _AXPY is tested in its 32-bit (SAXPY) and 64-bit (DAXPY) real forms; the second level BLAS routine _GEMV is tested in its 32-bit (SGEMV) and 64-bit (DGEMV) real forms; and the third level BLAS routine (_GEMM) is tested in its 32-bit (SGEMM) and 64-bit (DGEMM) real forms. The driver is written in Fortran 90 and the code must be linked to the vendor's native version of the BLAS for optimal performance. Each of the BLAS is tested for vectors or arrays with size of a power of 2 starting at 2.

5.1.5.1.4 ScaLAPACK Tests

A survey of the CTA leaders showed that a number of important projects use ScaLAPACK to solve dense linear systems and eigenvalue problems; therefore, we have included a test of the parallel performance of the ScaLAPACK asymmetric matrix Gaussian Elimination solver PDGESV, and the asymmetric matrix reduction to Hessenberg form routine PDGEHRD used to find eigenvalues.

Both the LU solver and the Hessenberg Reduction solve the following problem set:

- 1024 x 1024 matrix on a 2 x 8 processor grid (16 PEs)
- 2048 x 2048 matrix on a 2 x 16 processor grid (32 PEs)
- 4096 x 4096 matrix on a 2 x 32 processor grid (64 PEs)

- 8192 x 8192 matrix on a 2 x 64 processor grid (128 PEs)
- 16384 x 16384 matrix on a 2 x 128 processor grid (256 PEs)
- 32768 x 32768 matrix on a 2 x 256 processor grid (512 PEs).

5.1.5.1.5 Parallel Conjugate Gradient Solver Test

We have coded a parallel Conjugate Gradient solver in C to solve Laplace's equation on a cubic grid. Each test performs 1000 CG iterations. We solve the following problem sets:

- 3d Laplacian on 2 x 2 x 4 processor grid (16 PEs)
- 3d Laplacian on 2 x 4 x 4 processor grid (32 PEs)
- 3d Laplacian on 4 x 4 x 4 processor grid (64 PEs)
- 3d Laplacian on 8 x 4 x 4 processor grid (128 PEs)
- 3d Laplacian on 8 x 8 x 4 processor grid (256 PEs)
- 3d Laplacian on 8 x 8 x 8 processor grid (512 PEs)

5.1.5.2 Requirements for the CPU tests

ANSI C and Fortran 90 compilers, POSIX compliant, are required.

5.1.5.3 Resource Estimates for the CPU tests

All the tests may require 256 MBytes of memory for each processor. The BLAS test requires adding a search path to the native BLAS on the platform. The ScaLAPACK test requires adding a search path to ScaLAPACK or its equivalent in some library, and the ScaLAPACK test has been written with a minimum of miscellaneous routines

For portability, the ScaLAPACK test and the Conjugate Gradient test may require a search path to link the MPI library. The Fortran 90 coded tests use the POSIX standard wall-clock timing routine "get_system_clock" and the C coded tests use the C-callable routine "get_time_of_day."

The tests in cpubench will require about 2.5 hours to run.

5.1.5.4 Running the CPU tests

This tar file will create a directory called cpubench which will contain a README file, the Fortran 90 and C source code, two header files, a Makefile (and a GNU makefile if you use GNU's make).

In the Makefile you will find sections for Linux, Cray, IBM, SGI, Compaq, and Sun compiler flags and library search paths. Uncomment the section which is appropriate and edit it for optimal performance on your platform, or if necessary add a new section.

If you use GNU's make (gmake) edit the appropriate section of the file "cmplrflags.mk."

To make all of the cpubench test enter the command: make

You will find a subdirectory of cpubench called scripts. It contains three simple interactive run scripts called "do_cpu1", "do_cpu2", and "do_cpu3." You will have to edit all three scripts to edit the line, which does a cd to the working directory.

The first script "do_cpu1" will create a subdirectory called "results" (if it does not exist), and run all of the single-CPU tests (which consists of 136 tests).

The second script "do_cpu2" will execute the ScaLAPACK test, using a command called "RUNCMD." You may have to add an appropriate form of this command such as poe, mpirun, pam, mprun, etc.

The third script "do_cpu3" will execute the Conjugate Gradient solver, and you may have to define the variable RUNCMD there appropriately as well for your platform.

After you have run all the tests, enter the command: make tar

This will create a tar file called "cpubench.tar", which is the deliverable for cpubench. It will contain the source code and the results files, which are used to evaluate the benchmark.

5.1.5.5 Results for the CPU tests

All results are written to the results subdirectory. For instance, test fd101 is written to cputest/results/fd101.out. The results for all tests consist of a size (called n), a wall-clock time, given in seconds, and a MFLOPs count, based upon estimating the number of flops in each test and dividing by the wall-clock time. The ScaLAPACK test reports only the wall-time and the MFLOPs count, and the Conjugate Gradient test reports the wall-clock time, the MFLOPs, and the number of CG iterations and the norm of the final residual.

5.2 Instructions for Application Tests

Seven (7) actual applications were chosen this year based on expected usage and algorithmic considerations. The applications are AERO, Cobalt-60, RF-CTH, GAMESS, HYCOM, NAMD, and Electromagnetic Solver.

5.2.1 Technical Point of Contact for Application Tests

William A. Ward Jr.
Computer Sciences Corp

Voice: (601) 634-2512
Fax: (601) 634-3808
E-mail: William.A.Ward.Jr@erdc.usace.army.mil

5.2.2 Application Test Overview

This section provides guidance to vendors for preparing, conducting, and documenting the application portion of the HPCMP TI-04 benchmark test. Vendors may elect to run only a subset of the application tests. The Government-provided directory structure described below will provide the testing framework. Vendors shall augment this electronically supplied framework with test results and documentation and then return it to the HPCMPO / Government.

An important part of the test framework is contained in the *./ded* directory. In this directory is a subdirectory for each application. In each application subdirectory is a subdirectory for each test case (input data set). In each test case subdirectory is a subdirectory to contain test results for a particular number of processors. For example, the directory *./ded/gamess/standard* is the directory for the standard test case of GAMESS and it contains subdirectories names *016*, *032*, *048*, and *064*. In each of these processor count subdirectories are sample batch and submit scripts. These latter are automatically generated from **m4** template files. Vendors are strongly encouraged to use the template files to create their batch and submit scripts. From the vendor perspective, this will reduce the work required to create the scripts and aid in conformance to the Government's reporting requirements. Even after the scripts are (re)generated, it is permissible to manually modify them. However, the Government has endeavored to minimize the necessity of such modifications. As has already been noted, vendors may select the processor counts most appropriate for their proposed system.

A few words about the **m4** macro processor are appropriate at this point. **m4** is a standard UNIX utility. As in the C preprocessor, to which it is similar in function, and unlike the shells, macro variable names do not require a leading dollar sign. It is more powerful than the C preprocessor in the areas of arithmetic and conditionals.

5.2.3 Directory Structure

The test framework is a directory hierarchy used to supply vendors with guidance in conducting the tests. This guidance will include (a) either actual source programs or directions for obtaining such, (b) input data from example runs, (c) scripts and directions for executing the programs, and (d) output data to check the results. The directory hierarchy is described below.

./README is a text copy of this document (this is the definitive version).

./TAR produces the tar file of benchmark results to be returned to the government. This file may be edited to include additional files; *do not return input files, reference files, unmodified source files, or binary executables.*

./app contains one subdirectory for each application code/kernel. Source code and executables reside here. Source code for CTH, GAMESS, and NAMD must be obtained from the code owners as described in the *README* files for those applications.

./app/<APP> where *<APP>* is an application name, e.g., *./app/cth*. This directory contains the source distribution of application *<APP>*. The files/subdirectories in these directories will vary from one application to the next.

./app/<APP>/README contains documentation for installing and executing application *<APP>*.

./ded contains one subdirectory for each application. Results of the dedicated application tests reside here. It is important that vendors conform to the structure and use of this particular directory.

./ded/Build is the primary Bourne shell script for (re)building subdirectories for test results; vendors will need to modify this script if they elect to run only certain application tests or if they change the numbers of processors used in some of the tests.

./ded/Clean is a Bourne shell script for cleaning test result subdirectories. Depending on the modifications to *Build*, vendors may need to modify this script as well.

./ded/Defs.m4 is an input file to the **m4** macro processor; the definitions within this file may need to be modified.

./ded/Jobbat.m4 is an input file to the **m4** macro processor. This is the master template for all of the batch files; it must be modified as appropriate for a particular scheduler. The supplied example works for PBS on an SGI Origin 3000. Commands common to all scripts, e.g., setting certain environment variables should be placed here. As part of the script generation process, this file includes a *Jobbat.h* file described below that is specific to a particular application. The echo commands are part of the Government's reporting requirements and must not be deleted.

./ded/Jobsub.m4 is an input file to the m4 macro processor. This is the master template for all of the submit files; it must be modified to include the correct queue submission command. Note that the resulting submit file creates a ``*.qdate*" file that is used by the ``*.bat*" file.

./ded/Makefile is the master makefile for the test directory and script generation process; it is unlikely that this file will have to be modified.

*./ded/Makefile.** are other makefiles that are automatically installed in the directory hierarchy as part of the script generation process. Again, it is unlikely that these files will have to be modified.

./ded/<APP> is the top-level directory for tests results from application *<APP>*; e.g., *./ded/gamess*.

./ded/<APP>/Jobbat.h contains commands for executing this particular application. At script generation time (not script execution time), this file is included in the *Jobbat.m4* file noted above and various macro variables are expanded. In some cases, the command to execute a program in parallel, e.g., **mpirun** or **poe**, is embedded in this file. In other cases this file invokes a script containing the execute command; typically, such a script is somewhere in the *./app/<APP>* directory and is supplied either by the Government benchmark team or the code developer. Regardless of its location, it must be modified to suit a particular vendor's parallel operating environment.

./ded/<APP>/Jobsub.h contains commands to be performed before submitting a job for this particular application. At script generation time (not script execution time), this file is included in the *Jobbat.m4* file noted above and various macro variables are expanded. In many cases no such commands are necessary, and so the file is not present.

./ded/<APP>/Makefile is an automatically installed makefile and should not require modification.

./ded/<APP>/<CASE> is the top-level directory for all runs of test case *<CASE>* of application *<APP>*; e.g., *./ded/gamess/cycl*.

./ded/<APP>/<CASE>/Makefile is an automatically installed makefile and should not require modification.

./ded/<APP>/<CASE>/<NP> is the lowest level test case directory; e.g., *./ded/gamess/cycl/024*. It will contain automatically generated ``*.bat*" and ``*.sub*" files. Actual test runs should be performed in these directories, and output files preserved according to the directions in the *README* files for the various applications. This directory and all of its contents will be destroyed when the *Clean* script is executed.

./ded/<APP>/<CASE>/<NP>/.bat* is the automatically generated batch file for this particular run; it may be manually edited, but if the *Clean* script is executed, all changes will be lost.

5.2.4 Directions for Application Tests

1. Install each application code <*APP*> by following the instructions in the *./app/<APP>/README* file.
2. Modify the *./ded/Jobbat.m4* file as necessary to conform to a particular hardware/software/scheduler configuration.
3. Modify the *./ded/Jobsub.m4* file as necessary.
4. Determine the location of the command invoking the parallel operating environment, whether in *Jobbat.h* or in a file in the *./app* directory and change it to the appropriate command.
5. Execute the *Clean* command within the *./ded* directory.
6. Edit the *Build* command to generate scripts only for the applications and processor counts selected. Extraneous lines may be deleted or commented out. After completing these changes, execute *Build* within the *./ded* directory.
7. If necessary, make manual modifications to the individual batch and submit files. If all else fails, edit the original samples directly.
8. Submit the batch file using the submit file from each *./ded/<APP>/<CASE>/<NP>* directory to complete the test. You may rerun the script as necessary, but it is not advisable to have more than one job active in a given directory at the same time.

5.2.5 Application Test Cases

Vendors are invited to provide benchmark timings for the complete set of application test cases. However, vendors are also encouraged to submit results of some subset of the application test cases if the complete set cannot be run. Overall HPC system solutions that are optimized for subsets of DoD application codes will be considered. These runs must be performed in dedicated mode; i.e., with no other jobs running on the system. No interactive jobs are allowed.

5.2.6 Required Documentation

Many files will be generated during each run. Most of these files, including copies of the executables and input files, may be deleted after the run concludes. However, the following files must be preserved:

1. The script used to submit the batch file to the batch queue; this is the ``.*sub*" file discussed above.
2. The batch file so submitted; this is the ``.*bat*" file discussed above.
3. Any output written to standard output and standard error; these should be combined into a single file with suffix ``.*out*". Note that the output of the **echo** and **ls** commands embedded in the automatically generated ``.*bat*" file must appear in this listing.
4. Any other files as specified in the application's *README* file.

The performance data for each run must be filled in the table in Appendix A. All times must be reported in seconds. It must be possible for the Government to verify the results in this table by examining program output. Specifically, the elapsed wall times reported in the table must be calculated from the time stamps produced by the **echo** statements embedded in the ``*.bat*'' files. Vendors are welcome to provide other data regarding job and system performance if they so choose.

Finally, each vendor shall provide complete hardware and software specifications of the system. This specification includes, but is not limited to, the number of CPUs, the clock speed in MHz of each CPU, the number of front-end and batch nodes (if applicable), the total system memory in GBytes and how it is configured (local, NUMA, global, etc.), the total disk storage in GBytes and how it is configured (local, global, etc.), operating system compiler versions, and a description of the architecture of the CPU, memory hierarchy, and overall system. This latter item may include descriptions of such features as pipeline structure, dynamic instruction scheduling capabilities, cache sizes, cache coherence mechanisms, interconnects between processors, subgroupings of processors into symmetric multiprocessors, and software support for clustering, to name but a few. If the vendor chooses to supply some or all of this description in the form of a technical report or white paper, the Government will accept a PDF document instead of text.

Table of files to return

File to be returned	**Comment**
Synthetic tar file	Contains results of all synthetic benchmarks and directories, subdirectories.
Application benchmark results in softcopy	Results of application results in Excel spreadsheet
Application benchmark results in hardcopy	Hardcopy of all application results (section 6.2)
Configuration tables in softcopy	Softcopy of all configuration tables (Section 6.1)
Configuration tables in hardcopy	Hardcopy of all configuration tables. (Section 6.1)
Patch list	Patches and their descriptions
List of changes to Application Programs, Synthetic benchmarks, Make files, and run scripts	If none, state none, see section 4.7, and synthetic sections
Instructions for building and installing the test configurations	
Extrapolations for CPUBench and NETBench out to 512 processors, unless you have a benchmark configuration of 512 processors (or larger)	Use separate document.

Three sets of data should be created, and one set should be sent to each of the following locations:

DoD High Performance Computing Office
Attn: Contracting Officer
1110 N. Glebe Rd., Suite 650
Arlington VA 22201
(703) 812-8205

Instrumental Inc
Attn: Kim Payne
2748 E 82nd St
Bloomington, MN 55425-1365
(952) 345-2822

William A. Ward Jr.
C/o Ms. Daffney Wells
Computer Sciences Corp
3530 Manor Drive, Suite 4
Vicksburg, MS 39180
(601) 634-2512

6. Appendix A: Required Table submissions and formats

The timing for the Application benchmarks should be provided in softcopy and hardcopy format. The softcopy format is provided in the MS Excel spreadsheet named application_bm_results.xls. There are two workbooks in this spreadsheet. Tabs at the bottom of the spreadsheet access the workbooks. For these application benchmarks there are the following tables:

1. Edit, compile and link time table Note: Please include timing for each application that is built including applications that require multiple builds for each grid size and performance.
2. Performance benchmark timing table.

6.1 Configuration Tables

The following configuration tables are to be filled out in both softcopy and hardcopy. These tables are provided in an attachment named configuration_tables.doc.

CPU Table

CPU	Benchmark System	Proposed System
What is the CPU clock rate?		
What is the CPU instruction issue rate?		

	What is the number of floating point units?		
	What is the number of integer units?		
	What is the size and type of the internal caches?		
	What is the maximum SMP CPUs?		
	How many hardware counters are supported and what is the number of counter that can be accessed per run?		

Memory Table

	Main Memory	Benchmark System	Proposed System
	What is the maximum main memory performance for a single CPU?		
	What is the total main memory performance to all CPUs?		
	Is bandwidth and latency consistent across all CPUs and all of memory (If not please explain difference and techniques to minimize latency. E.g. memory placement issues)?		
	List the supported memory size(s)		

How many total memory banks/sections are there?		

Cache Memory	Benchmark System	Proposed System
What is the bandwidth to cache memory from a single CPU and multiple CPUs if they share the same cache?		
What is the startup latency for cache?		
What is the cache coherency bandwidth?		
What is the size of the cache line?		
What is the cache to main memory bandwidth per cache and for all caches?		
What are the cache line sizes for all caches?		

System Table

OS Questions	Benchmark System	Proposed System
What is the size of the system page for standard user applications		
Does TCP/IP stack use DMA transfers (e.g. Zero Copy TCP)?		
What is the maximum number of CPUs supported by the OS?		
What is the maximum LUN size supported?		
What is the maximum supported memory size for a single application?		
What is the largest I/O request supported?		
Is direct I/O supported; if so, provide an explanation of how to use direct I/O?		
What batch scheduler used in benchmark?		
File System/Volume Manager Questions (if multiple file systems and/or Volume managers are proposed these questions must be answered for each file system)	**Benchmark System**	**Proposed System**
Does proposed file system and/or volume manager separate data and meta data?		

	Does the proposed file system support round-robin file allocation instead of striping all files across all devices?		
	What is the largest file system supported and what is the largest file system tested?		
	Does the file system support preallocation, if so how?		
	What is the largest supported file system block size, or allocation unit?		
	What is the largest single file supported		
	What is the largest I/O request to system supported		
	What is the largest I/O request to file system supported		

Switch Table

	Switch Questions (if switches are proposed)	Benchmark System	Proposed System
	What is the performance from any port to any port in MB/sec?		
	What is the internal design?		
	a. Port density per board		
	b. Bus configuration		
	c. Backplane configuration		
	How many ISL connections are supported?		
	Is translative mode supported for loop devices?		
	How many buffer credits dynamically allocated to a port?		

How many buffer credits are statically allocated to a port?		
What HBAs are supported?		
What tapes are supported?		
What RAIDs are supported?		
What is the mechanism for port security?		
What is the quantity of security zones (e.g. can a zone be controlled and managed by a group? If so, how many?)		
What is the maximum supported switch to switch distance?		
What is the maximum supported switch to devicedistance?		
What is the board to board latency?		
What is the interboard latency?		
What is the total backplane bandwidth?		

Communication Table

Bus Questions	Benchmark System	Proposed System
Is the proposed bus PCI or PCI-X (state speed and width)?		
Does the bus run full duplex at the specified rates (please provide test data showing full duplex rates)?		

HBA Questions	Benchmark System	Proposed System
How many ports per HBA?		

What is the rated speed of the HBA?		
What is the size of the HBA command queue?		
What is the full duplex performance of the HBA (provide actual test data and explain)?		

LAN support	Benchmark System	Proposed System
What are the network interconnects supported (list all)?		
What is the compatibility information for each interface?		
What are the MTU sizes for each interface?		
What is the full duplex performance and CPU overhead for each interface?		
Is trunking supported and with what interfaces and switches?		

RAID Table

RAID Vendor Questions	Benchmark System	Proposed System
What is the chipset used?		
What is the internal bus type and design, PCI or PCI-X		

	Please provide cache hardware information and size		
	What is the bandwidth for write cache mirror?		
	What is the bandwidth to disk?		
	What is the processor type and speed?		
	Explain how dual active/passive support work for controllers, power, and cache		
	What LUN count is supported by controller		
	How many hosts can be connected per controller?		
	What is the loop count controller to disk?		
	What is the load balancing controller to disk?		
	What disk types supported?		
	What is the RAID configuration (e.g. RAID 1,5, 0+1 etc)?		
	What is the controller failover support; active or passive?		
	LUN size. Please discuss number of TB per LUN supported per OS (Solaris, Win2K, Tru64, HP-UX, AIX)		
	Are the devices chained? If so, how many are on a loop/bus?		

	How do backend loops support the LUNs' load balancing?		
	How do backend loops address availability of data?		
	Is remote copy supported?		
	Provide MTBF/MTTR/MTDL for all supported RAID levels		
	What components can be upgraded hot (both hardware and software)?		
	Provide alignment/block size values supported		
	Provide cache allocations supported		
	System settings (provide tunable information)		
	For both RAID 1 & 5 (and any others that support redundancy). We want to know how write reconstruct works both when there are hot spares and when there are not.		
	What HBAs are supported and tested?		
	What switches supported and tested?		
	Can the blocksize/alignment be changed without taking system off-line? If so explain limitations.		
	What is the command queue size?		

	What is the maximum OPS to disk (please provide hardware and software configuration and benchmark test environment including OS, software used, HBAs and firmware releases)?		
	What is the maximum OPS to cache (please provide hardware and software configuration and benchmark test environment including OS, software used, HBAs and firmware releases)?		
	What is the maximum streaming to disk (please provide hardware and software configuration and benchmark test environment including OS, software used, HBAs and firmware releases)?		
	Provide information in above 3 questions with and without write cache mirror if supported		
	What are the cache tunables (please provide all information)?		
	What is the caching algorithm (separate read/write cache, high water marks etc)?		

6.2 Application Benchmark Results

The application benchmark results should be provided in the included spreadsheet named application_bm_results.xls. A copy of the table is included here.

Code	Test Case	CPUs	NAVO IBM POWER4 Marcellus 1184
AERO	**std**		
		1	16502
		2	***10943***
		3	9483
		4	8651
		5	8300
		6	8066
		7	7848
		8	7336
Cobalt-60	**std**		
		16	8337
		32	3733
		48	2409
		64	***1725***
		128	770
		256	384
Cobalt-60	**lg**		
		32	38353
		64	17327
		128	8170
		192	5190
		256	***3736***
		384	2471
		512	1792

GAMESS	**std**		
		16	6774
		32	3803
		48	2798
		64	***2320***
GAMESS	**lg**		
		64	11805
		128	6788
		192	4932
		256	***4038***
		384	3245
		512	2828
HYCOM	**std**		
		47	2344
		59	***1799***
		80	1392
		96	1210
		111	1080
		124	974
HYCOM	**lg**		
		234	4109
		256	***3877***
		314	3272
NAMD	**std**		
		16	4002
		32	2381
		48	1945
		64	***1773***

NAMD	**lg**		
		108	6208
		216	***3843***
		324	2958
		432	2473
O-O-Core	**std**		
		16	10416
		32	4913
		48	3035
		64	***2395***
O-O-Core	**lg**		
		128	9125
		256	***4022***
		384	3390
		512	2744
RFCTH	**std**		
		16	4420
		32	2492
		48	1983
		64	***1529***
RFCTH	**lg**		
		128	6669
		256	***4852***
		384	3928
		512	3074